Bleeding Edge Press

KONG

BECOMING A KING OF API GATEWAYS

Alex Kovalevych, Robert Buchanan,
Daniel Lee, Chelsy Mooy, Xavier Bruhiere
& Jose Ramon Huerga

Kong: Becoming a King of API Gateways

By Alex Kovalevych, Robert Buchanan, Daniel Lee, Chelsy Mooy, Xavier Bruhiere, and Jose Ramon Huerga

Kong: Becoming a King of API Gateways

ISBN: 9781939902511

Published by: Bleeding Edge Press, Santa Rosa, CA 95404

Title: Kong: Becoming a King of API Gateways

Authors: Alex Kovalevych, Robert Buchanan, Daniel Lee, Chelsy Mooy, Xavier Bruhiere, and Jose Ramon Huerga

Acquisitions Editor: Christina Rudloff

Editors: Troy Mott

Website: bleedingedgepress.com

Table of Contents

Preface

Who is this book for?

This book is useful for IT architects, DevOps engineers, CTOs and security experts willing to understand how to use Kong to create and expose APIs.

Even if you are not already familiar with Kong, it will only take a few minutes to create your first API.

What do you need to know prior to reading?

You don't need to know Kong to read this book! You only need to have a basic understanding of REST, JSON and HTTP, but you don't need an in-depth knowledge because Kong's mission is to provide easy API publishing.

You will also need a modern browser: Google Chrome, Mozilla Firefox, Microsoft Edge or Apple Safari.

The online example

All of the code for the sample project in this book can be found at:

https://github.com/backstopmedia/kong-book-example

What will this book provide?

By the end of this book, you will understand how to:

- Use an API gateway to simplify and improve the security of your microservices architecture
- Write Kong plugins with Lua
- Deploy Kong and Cassandra in a multi-region environment
- Use load balancing features.

Book co-authors and contributors

Alex Kovalevych is an experienced web developer from Kyiv, Ukraine. Having degrees in computer science and finance, he is a polyglot web developer, constantly learning and using cutting edge technologies. He has been working on different worldwide projects in gaming and health fields.

Chelsy Mooy has been handling multiple SaaS for Finance projects as a CTO and Co-Founder of PT Thunderlabs Indonesia. She believes that well-constructed technology is a powerful “bunshin no jutsu." She lives in Malang, Indonesia with “the dog” and wears a skirt on Sunday.

Daniel Lee is a skilled engineer and is enthusiastic about tech. He really loves data science, cloud engineering, and service development. He was a former chief developer at the web community Yourssu, and now he is running a legal tech startup, Dbrain Science, in South Korea.

Robert Buchanan is a passionate, forever learning software craftsman that picks the right tool for the right job. He is a polyglot engineer who has been in all aspects of the cycle as a UI developer, service developer, cloud engineer and performance analyst. He lives with his wife and kid in Northern Kentucky, United States.

Xavier Bruhiere is a senior data engineer growing the Kpler team in Singapore. He has been crunching data since founding his own quantitative investment company and through startups in Paris. He loves to collaborate with smart people, visiting new countries and eating Italian delights.

Jose Ramon Huerga Ayuso is an API Management and Microservices expert. During the 80s he was a teenage programmer and later in the 90s he earned a degree in computer science. He has been working in multiple fields, including SOA, QA, ECM, and CRM. He lives with his wife and two kids in Madrid, Spain.

Technical Reviewers

We would like to the thank the following technical reviewers for their early feedback and generous, careful critiques: Nicolas Huray and Michallis Pashidis. Thanks also to Eber Herrera, Panagis Tselentis, and Damian Czaja.

API Gateway Techniques 1

The API Gateway is a component of API Management.

API Management takes a bigger scope than API Gateway, which means API Management is the whole process of generating and publishing APIs, while API Gateway is more like an API interface or middleware. In practice, the API Gateway works as a connector between services that are responsible to receive requests, pass them to the backend service and give a response back to consumers. Hence, the API Gateway must be able to be identified. It is also mean to be as efficient as possible to work effectively.

All of the code for the sample project in this book can be found **here**.

Multi Consumer management

As written in Chapter 3, one of the API Gateway uses is to control traffic. API Traffic depends on how many consumers and how many requests per each consumer. It is important to differentiate which consumer is prohibited to access your APIs, and what consumer can access that. Back to the cinema system example. Let's say there's two third party interfaces for a payment gateway, from bank A and bank B. Bank A can only update the transactions status of their customers that use their services, and so does bank B. It will be troublesome if bank A is able to update transaction statuses of customers from bank B.

Versioning management

The consumers of an API will invest a significant amount of time and resources building client applications. It is important to guarantee that the API will provide its service for a long period of time without breaking the commitments undertaken with the consumer organizations. If an API is modified while client applications are using the old version, it must be verified beforehand so the new features will be able to coexist with the existing functionality.

In terms of the API Gateway itself, the benefit of using versioning is to help your system to be upgraded smoothly. Take an example of your Android native app. You have released version 2.0.0 where there is a payment feature. Then you upgraded your app having version 2.1.0 that provides push notifications about payment statuses. Say that your version 2.1.0 did not work in Android kit-kat below. If you differentiate your consumers based on versions there won't be any problems about unnecessary data for Android kit-kat or previous versions of Android. You can always send data based on consumers' need.

The next paragraphs detail which types of changes are backward compatible, which numbering conventions apply, and how to make Kong handle several versions of an API on the same server.

CHANGES THAT MAINTAIN BACKWARD COMPATIBILITY

The following changes on an API are backward compatible:

- Add a resyource (path) or a verb (operation) to an API.
- Add optional input fields (query or path parameters, headers) to existing operations.
- Convert an input parameter from mandatory to optional.
- Add a new output parameter in the response.

CHANGES THAT BREAK BACKWARD COMPATIBILITY

These changes will break backward compatibility:

- Remove resyources (path) or verbs (operations).
- Add mandatory input fields.
- Convert input fields from optional to mandatory.
- Modify the name or the type of a parameter.
- Add pagination support to a query that returns multiple objects. This will affect to existing client applications, as these applications will not be aware that a query is now returning only a page of the results and not the complete result set.

MAJOR, MINOR AND FIX VERSIONS

Typically an API version will be identified with three numbers separated by a dot: MAJOR.MINOR.PATCH. The following rules explain when a new version of an API should be marked as a MAJOR change, a MINOR change or just a PATCH:

- MAJOR. A new MAJOR version will be used when a new release includes incompatible changes to the API.
- MINOR. A new MINOR version will be used when new features are included to the API and these changes are backward compatible.
- PATCH. A new PATCH version will be used when just some bug fixes are done in the API, obviously maintaining the backward compatibility.

SUPPORTING SEVERAL VERSIONS ON THE SAME GATEWAY

Currently, Kong supports the existence of different versions of an API, provided that the version number is included in the URL:

- Requests to **http://localhost:8000/v1/movies** would send the request to the microservice that implements the v1 of the functionality.
- Requests to **http://localhost:8000/v2/movies** would send the request to v2.

Logging for failure or error

API Gateway allow us to store logs of requests and responses using various transports, like TCP, UDP, HTTP, and others. Log files help us to analyze traffic and make it easier to trace errors or fails inside the whole system. Logging for failure or error handling in the Gateway level must be done carefully, since it comes from various services and consumers.

CLEAR DATA ORIGIN

It is important to know which services or consumers produced errors or failures. The sequences will be:

- Which service or consumer
- When it happens
- Which event it is
- Who triggered the event (authorized user if any)
- What is the processed data (both input and output)
- What is the expected input / output

Both services and consumers have to store local log files. It helps developer to trace wether error or failure happen in the gateway level or in the services level or even consumers level.

NOTIFICATIONS FOR FAILURES OR ERRORS

Besides the activity of storing local errors log, being alert about failure or error before your costumer finds out is an important task. There are several plugins for traffic alerts in Kong that can be used, like Runscope for HTTP request or response.

Caching criteria

Caching process happen in-memory because the need of avoid querying to datastore every time. But not all requests can be cached. For example, available movies. Imagine your customer see “Kong : Micro Island” movies showing at 11:45 and there was available seat for her. She bought a ticket, but a pop up message appeared saying “There is no seat available,” while the movies list shows that there is an available seat. It won’t be a nice experience for customers. The criteria of caching will be:

- Time based expired. This one can be used to cache the data that did not frequently change, like casts, titles, or synopsis
- Data frequency. This one can be used for data that frequently changes, but is also frequently accessed, like ratings.

Rate limiting

Providing an API Gateway for various consumers means that your API Gateway should be available and reliable in any circumstances. There are scenarios that might happen in the API Gateway usage:

- Infinite requests from one consumer within a second that caused the API Gateway to be too busy to handle other requests.
- You need to keep it fair between high priority requests and low priority requests.

Rate limiting helps cover both scenarios in many ways.

LIMIT THE AMOUNT OF REQUESTS CONSUMERS MAY HAVE

In order to protect the API Gateway from bottlenecks, it is better to prevent the requests from being made from one consumer. Here, you can set the number of allowed requests based on a period of time. It is better to decrease the number of potential unused requests. In your responsive web app, where showtime lists displayed on the frontpage and also the most frequent accessed page, you can set for 60 requests per second. As for movie details, you can set 30 requests per second, as it is less accessed.

ALLOCATE MORE FOR HIGHER PRIORITY REQUESTS

Which is more fatal, terminating transactions and leaving incomplete payments, or asking customer to reload their homepage?

Always put finance transactions as a higher priority than displaying processes. It is better to ask customers to reload their homepage instead of leaving an incomplete finance transaction. There is the option of using redis or cluster. But once again you have to wisely choose which data should be using redis or cluster since it is done in-memory.

Routing Common Issues

To keep your services dependent, can you do nested API Gateway?

The answer should be an easy no. But you often fail in this option and make the API Gateway do requests for itself. The reason for not doing nested requests is the potential of an infinite loop if you don't carefully map a request path. In the other hand, you left your API Gateway dependent on itself for results.

HTTP CORS issues

If your API Gateway makes an HTTP request to a different domain, it needs to be CORS-friendly. If you use cross domain requests, you need to add the Access-Control-Allow-Origin header to your responses in your handlers. If you're using cookies or other authentication, you'll also need to add the Access-Control-Allow-Credentials header to your response. But no worries, Kong has a cors plugin so you don't have to manually doing this.

Specific-Purpose route

Is it okay to have two routes for one request in a different scenario? For example, you have a route to display showtime lists that display price in the Euro and another route to display showtime lists that display price in USD.

If you do routes like that, it is the same as putting business logic in the frontend level. Which means, both the native Android app and responsive web app should have known where to call and when. Both of them have to detect which nationality belongs to a customer and which route should be called.

POST OR GET NOT ANY FOR HTTP REQUESTS

The basic HTTP methods for REST APIs, with no ambiguity, will be helpful for data logging. Here are those methods:

- GET—For returning resources
- POST—For creating a new resource
- PUT—For replacing a resource
- PATCH—For updating a resource with versions
- DELETE—For deleting a resource

Business authorization in the API Gateway

The API Gateway may authenticate the user. But this does not mean it can always authorize users. What if a blacklist user can't book a movie show, but can see a movie showtime? Should it be authorized in the Gateway level? A user's scope can be very specific and changes the core of the system. It is better to put a user's authorization, especially business authorization for a user in level of services. There are several options of access tokens can be passed to services from API Gateway like OAuth2. Chapter 10 will describe this.

Composing data in the API Gateway

In order to make responses more consumer-oriented, you use transformers. Your scenarios of using a transformer might be like these:

1. Collaborating two or more services respond in a single request
2. Remove, rename, replace, add, or even append a request and/or response

Let's say your native Android app runs in a medium screen size capable of displaying information of movie showtimes and details like casts and ratings. And you construct your services like this:

1. Showtime services

ID	TITLE	SHOWTIME	PRICE	Short Desc
MV-1709-001	Kong: Micro Island	12:45	$10	...
MV-1709-001	Alien: Components	13:45	$10	...
MV-1709-003	King Artie: Legend of the Swarm	21:15	$10	...

1. Movie services

ID	TITLE	CASTS	RATINGS	RELEASE DATE
MV-1709-001	Kong: Micro Island	Brit, Tim, Sam	8	2017-09-01
MV-1709-001	Alien: Components	Ailee, Dave	7	2017-09-01
MV-1709-003	King Artie: Legend of the Swarm	Cliff, Joan	8	2017-09-01

And your Native android app needs data like this:

ID	TITLE	SHOWTIME	PRICE	CASTS	RATINGS
MV-1709-001	Kong: Micro Island	12:45	$10	Brit, Tim, Sam	8
MV-1709-001	Alien: Components	13:45	$10	Ailee, Dave	7
MV-1709-003	King Artie: Legend of the Swarm	21:15	$10	Cliff, Joan	8

To have results like in the table above, you need to join your showtime services and movie services based on a movie ID. It can be done in the Gateway level. As you know, most of the logical functions in the Gateway level were done in-memory. Again, it costs more for the API Gateway. Besides, the API Gateway must be smart enough to handle unstructured data since the Microservices architecture allows us to have various databases, like mongoDB, mySQL, Cassandra and others. The other option is applying the Command Query Responsibility Segregation (CQRS) pattern to do the composing.

Transforming API requests and responses

Let's say you applied the CQRS Pattern and have read-only data like this:

ID	TITLE	SHOWTIME	PRICE	CASTS	RATINGS
MV-1709-001	Kong: Micro Island	12:45	$10	Brit, Tim, Sam	8
MV-1709-001	Alien: Components	13:45	$10	Ailee, Dave	7
MV-1709-003	King Artie: Legend of the Swarm	21:15	$10	Cliff, Joan	8

At the same time your cinema system is scaling up, allowing a display price based on users' nationality, and your app is available for smaller screen size so that there is no need to display ratings and casts.

ID	TITLE	SHOWTIME	PRICE
MV-1709-001	Kong: Micro Island	12:45	€8.06
MV-1709-001	Alien: Components	13:45	€8.06
MV-1709-003	King Artie: Legend of the Swarm	21:15	€8.06

Modifying responses for specific purposes are better done in the services level. Since handling policies in the Gateway level could mess up the application design. Besides, you have to change all consumers' settings if the policies have changed.

Though microservices allow every service to be deployed independently, it will be difficult if each service has different terminology of similar things. For example, doing pagination. If your showtime services doing pagination use `skip` and `take` as parameters, while your movie services use parameters like `page` and `take`. Then your API Gateway has to translate consumer's parameters so that can be understood by services.

Doing workflow in the API Gateway

You have a flow like this:

1. Customer books two tickets for watching a movie in a booking service
2. Service generates an invoice in the billing service
3. System sends an invoice to email using a third party mail service

It is common to have relationships between processes of different services like this sample above. But, it is better not to do it in the Gateway level. Despite the memory usage issue, it causes high dependencies between services in the Gateway level, and risks too much if one of the flows fails. Doing this in tje Gateway level is much more like doing monolith scenarios in the Gateway level. There are options to do event driven in a microservices way, that uses a message broker.

Summary

You should now have a deeper understanding of the API Gateway for Kong. In the next chapter we focus on API security.